CODEX CLI

Setup Reference Card

OpenAI's agentic command-line coding tool - Docs: github.com/openai/codex https://developers.openai.com/codex/cli

Codex CLI is OpenAI's terminal-based coding agent that reads and edits files, runs commands, and works directly inside your repository. This card covers connecting it to the American Science Cloud's **Model Access Gateway (MAG)**.

1 - Install

Codex is a standalone binary (built in Rust). Install with the platform script or a package manager - pick **one** method per machine.

Platform	Command	Updates
macOS / Linux / WSL	`curl -fsSL https://chatgpt.com/codex/install.sh | sh`	Re-run script
Windows (PowerShell)	`irm https://chatgpt.com/codex/install.ps1 | iex`	Re-run script
macOS (Homebrew)	`brew install --cask codex`	`brew upgrade`
Windows (WinGet)	`winget install OpenAI.Codex`	`winget upgrade`

Also available (cross-platform, needs Node.js): `npm install -g @openai/codex`

Verify the install - open a new terminal in your project, then:

```
codex --version       # confirm the binary is on your PATH
codex                 # start an interactive session
codex update          # Check for and apply an update when the installed release supports self-update
```

> **Prerequisite:** You need an AmSC account and a MAG PAT (Personal Access Token) before Codex can reach a model. Set that up in the step below.

2 - Connect to the Model Access Gateway (MAG)

Step 1 – Get an American Science Cloud (AmSC) Account

Follow the instructions listed here: https://amsc-docs-d762d2.gitlab.io/GM-getting-started#access--login to create an AmSC account. Once the account is approved, proceed to the next step.

Step 2 – Request MAG access and generate a Personal Access Token (PAT)

Follow the instructions listed here: https://amsc-docs-d762d2.gitlab.io/model-access-gateway#quick-start

Step 3 - Set environment variables

Add these to your shell profile (`~/.zshrc`, `~/.bash_profile`, or PowerShell `$PROFILE`). Codex reads the OpenAI-style variables.

```
# Your MAG project key
export AMSC_I2_API_KEY=<your-MAG-PAT>

# Map it to the OpenAI-compatible variables Codex expects
export OPENAI_API_KEY=$AMSC_I2_API_KEY
export OPENAI_BASE_URL=https://i2-api.genesis.american-science-cloud.org/
```

Step 3 - Verify the variables loaded

Restart your shell (or log out and back in), then check:

```
env | grep OPENAI      # OPENAI_API_KEY and OPENAI_BASE_URL should appear
```

3 - Configure Codex - `~/.codex/config.toml`

Codex reserves the built-in provider id `openai`, so define a **custom** MAG provider and point Codex at it. Create `~/.codex/config.toml`:

```
model_provider = "amsc"

[model_providers.amsc]
name = "AmSC Model Access Gateway"
base_url = "https://i2-api.genesis.american-science-cloud.org/v1"
env_key = "OPENAI_API_KEY"
```

4 - Start Codex

```
cd my-project
codex                          # interactive session in this repo
codex "explain the auth flow"       # run a one-off prompt
codex -m gpt-5.3-codex              # explicitly choose the model for this run
```

5 - Everyday Commands

Command	What it does
codex	Start an interactive session in the current directory
codex "..."	Run a one-off prompt, e.g. codex "add tests for utils.py"
codex -m <model>	Pick a model for this session (e.g. gpt-5.3-codex)
codex --profile <name>	Load a named config layer (~/.codex/<name>.config.toml)
codex exec "..."	Non-interactive / scripted run of a single task
codex --version	Print the installed Codex version
/	(TUI) List in-session slash commands
/status	(TUI) Display session configuration and token usage. Note, /usage will not work for API billing via a custom provider like the AmSC Model Access Gateway

6 - Test Your Connection

To confirm that the MAG is correctly configured and answers your requests before starting a long Codex session:

```
curl -s $OPENAI_BASE_URL/v1/chat/completions \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"model":"gpt-5.3-codex",
       "messages":[{"role":"user","content":"Reply with OK."}]}'
```

To sweep every model your key can reach, use the "**Test all models**" script in the MAG docs - it queries /model/info and pings each chat and embedding model.

Keep secrets out of source. Never paste a raw key into code, config committed to git, or a shared file. Keep it in an environment variable (or a git-ignored profile) and reference $OPENAI_API_KEY.